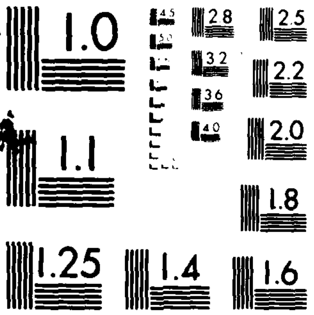
1.0
4.5
5.0
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6

PLAN SYNTHESIS:

A Logical Perspective

Stanley J. Rosenschein

Artificial Intelligence Center
SRI, International
333 Ravenswood Avenue
Menlo Park, CA 94025

March, 1981

Contract N00014-80-C-0296

ABSTRACT

This paper explores some theoretical issues of robot planning from the perspective of propositional dynamic logic. A generalized notion of "progression" and "regression" of conditions through actions is developed. This leads to a bidirectional single-level planning algorithm which is easily extended to hierarchical planning. Multiple pre/post-condition pairs, complex (conjunctive, disjunctive) goals, goals of maintenance and prevention, and plans with tests are all handled in a natural way. The logical framwork is used to clarify gaps in existing "nonlinear" and "hierachical" planning strategies.

i—

Table of Contents

1. Introduction

Although the connection between the A.I. planning problem and automated program synthesis is widely acknowledged, relatively little planning research has made explicit use of concepts from the logic of programs. Such logics, however, offer theoretical insight into various issues in A.I. planning, including compound goals and levels of abstraction. [11, 12, 14, 4] Many of these issues arise in their purest form in domains describable in the propositional calculus (e.g. simple blocks worlds) as evidenced by the literature on the subject. [13, 11, 15] Thus for clarity and continuity, we choose the propositional setting to develop a unified, abstract treatment of these issues using propositional dynamic logic (PDL) as our primary logical tool. [7, 5, 8, 9, 3]

PDL is a decidable modal propositional logic for reasoning about binary state-relations induced by programs. In theory, the existence of such a logic provides an immediate "solution" to the propositional planning problem: One could systematically substitute all possible plans into a schema (the specification) which asserts the desired property of the plan. The resulting expressions could be tested for validity to filter out non-solutions. Unfortunately, this fact is of little practical consequence, as such a procedure is certain to be grossly inefficient. The approach developed here imposes additional structure by (1) considering a class of problems that require, in effect, only non-modal reasoning and (2) using suitable "progression" and "regression" operators to structure the search for a solution and allow early pruning of hopeless paths.

Surprisingly, even our restricted formulation covers a more general class of problems than are handled by most comparable A.I. planning methods. For instance, we allow goals to be arbitrary wffs, so disjunctive goals (cited as an unsolved problem by Sacerdoti [12]) require no special treatment at all. The approach provides a theoretical basis for hierarchical plan generation that ties in directly with current ideas on hierarchical program development (see section 3.3). In addition, the use of program logic provides a formal basis for specifying and verifying the plan-generating system itself.

Although our work can be generalized along several dimensions (propositional axiom schemata, plans with loops, quantified pre/post-conditions, etc.), these are beyond the scope of the current paper, which focuses instead on the essential structure of the approach. At the same time, it should be noted that the use of axiom schemata seems to be a minimal requirement for a practical application. This paper should be regarded as a foundational study aimed at deepening our understanding of planning; a separate paper will describe implementation considerations. [10]

2. Preliminaries

This section briefly presents the basic concepts of a loop-free fragment of propositional dynamic logic (PDL). The interested reader is referred to [7, 5, 3] for a more thorough treatment of dynamic logic in its entirety. Though dynamic logic is ordinarily used to reason about _programs_, it is equally appropriate for reasoning about _plans_ (in the A.I. sense); thus in this paper the terms _program_ and _plan_ are used interchangeably.

2.1. Syntax

Let $\mathcal{P}$ and $\mathcal{A}$ denote two symbol sets: atomic propositions and atomic actions, respectively. Define wffs $P_{\mathcal{P},\mathcal{A}}$ and programs $A_{\mathcal{P},\mathcal{A}}$ simultaneously (deleting subscripts for convenience):

1. $\mathcal{P} \subseteq P$
2. $\mathcal{A} \subseteq A$
3. If $p, q \in P$ then $\neg p$, $p \vee q \in P$
4. If $p \in P$ and $\alpha \in A$ then $\langle\alpha\rangle p \in P$
5. $\Lambda \in A$
6. If $p \in P$ and p is nonmodal (see below) then $p? \in A$
7. If $\alpha, \beta \in A$ then $\alpha;\beta$, $\alpha \cup \beta \in A$

A formula is nonmodal if it contains no subformula of the form $\langle\alpha\rangle p$. We abbreviate $\neg(\neg p \vee \neg q)$ as $p \wedge q$, $\neg p \vee q$ as $p \supset q$, $(p \supset q) \wedge (q \supset p)$ as $p \equiv q$, $\neg\langle\alpha\rangle\neg p$ as $[\alpha]p$, $p \vee \neg p$ as true, and $p \wedge \neg p$ as false. Parentheses are used when necessary according to the usual conventions.

2.2. Semantics

A structure S is a triple (W,π,m) where W is a non-empty set of "worlds," $\pi: \mathcal{P} \to 2^W$, and $m: \mathcal{A} \to 2^{W \times W}$. That is, π assigns to each atomic proposition p the subset of W where p holds, and m assigns to each atomic action a the binary relation over W representing the next-state relation for a. Given a structure S, meanings can be assigned to arbitrary programs and formulae by extending m and π:

Meanings of Programs

1. $m(\Lambda) = \{(s,s) \mid s \in W\}$
   (identity relation over W)
2. $m(p?) = \{(s,s) \mid s \in \pi(p)\}$
   (identity relation restricted to worlds where p holds)
3. $m(\alpha;\beta) = m(\alpha) \circ m(\beta)$
   (composition of relations)
4. $m(\alpha \cup \beta) = m(\alpha) \cup m(\beta)$
   (union of relations considered as sets)

Meanings of Formulae

1. $\pi(\neg p) = W - \pi(p)$
2. $\pi(p \vee q) = \pi(p) \cup \pi(q)$
3. $\pi(\langle\alpha\rangle p) = \{ s \in W \mid \exists t \in W.\ (s,t) \in m(\alpha) \text{ and } t \in \pi(p) \}$

The last equation asserts that $\langle\alpha\rangle p$ is true in those worlds s from which another world t is reachable via α's next-state relation such that p holds in t. In general our formulas will involve the dual of $\langle\alpha\rangle$, namely $[\alpha]$. $[\alpha]p$ can be read "after α, p." The intent is that p holds in all worlds accessible via α.

A formula p is <u>valid in a structure</u> $S = (W,\pi,m)$ (written $S \models p$) iff $\pi(p) = W$; p is <u>valid</u> (written $\models p$) iff it is valid in every structure.

2.3. Axiomatics

The following system captures the semantics given in the previous section:

Axioms

1. Axioms of the propositional calculus
2. $[\alpha](p \supset q) \supset ([\alpha]p \supset [\alpha]q)$
3. $[\Lambda]p \equiv p$
4. $[p?]q \equiv p \supset q$
5. $[\alpha;\beta]p \equiv [\alpha][\beta]p$
6. $[\alpha \cup \beta]p \equiv [\alpha]p \wedge [\beta]p$

Rules of Inference

1. From p, $p \supset q$ derive q	(Modus Ponens)
2. From p derive $[\alpha]p$	(Necessitation)

If a formula p follows from these axioms under the stated rules of inference, we say it is <u>provable</u> and write $\vdash p$; if p can be proved from a set of assumptions, Q, we write $Q \vdash p$.

2.4. A Restricted Class of Programs

PDL breaks the ordinary conditional statement into more primitive notions of "test" (?) and "non-deterministic execution" ($\cup$). Though we allow the primitive actions to be non-deterministic, we shall only be interested in deterministic combining forms. Thus we limit the use of ? and $\cup$ to contexts of the form $(p?;\alpha) \cup (\neg p?;\beta)$ and require (for convenience only) that p be atomic. This corresponds to the ordinary conditional, so we abbreviate this program form to $p \rightarrow \alpha, \beta$ and call the class of programs obeying these syntactic restrictions <u>C-programs</u> (symbolically $\hat{A}_{\rho,\mathfrak{a}}$). The requirement that p be atomic is not restrictive, since arbitrary boolean combinations of tests can be expressed by appropriate use of (possibly nested) conditionals. For example, $\neg p \rightarrow \alpha, \beta$ is equivalent to $p \rightarrow \beta, \alpha$. $(p \wedge q) \rightarrow \alpha, \beta$ is equivalent to $(p \rightarrow (q \rightarrow \alpha, \beta), \beta)$, etc.

An important property of our combining forms is that they preserve termination; if the primitives always terminate, every C-program over those primitives will always terminate. (Loops are conspicuously absent.) In PDL, the fact that a program α always terminates is expressed $\langle\alpha\rangle$true.

3. A Planning Method

Having described a suitable language and logic, we are now in a position to discuss planning methods. Section 3.1 contains the formal definition of a (single-level) "planning problem" and the corresponding notion of a "solution". This leads directly (section 3.2.2) to a bidirectional (single-level) planning algorithm based on "progressing" and "regressing" conditions through actions. Section 3.3 describes how the hierarchical planning problem can be regarded as a succession of single-level problems in a way that makes the connection between the levels logically precise.

3.1. Definitions

A <u>planning problem</u> is a triple (V,Q,R(u)) where

$V = (\mathcal{P},\mathcal{A})$	is the <u>vocabulary</u> of the problem, consisting of the atomic propositions and the atomic actions.[1]
Q	is a finite set of axioms, which we will refer to as <u>domain constraints</u>. Q is partitioned into two subsets: static constraints, which are nonmodal formulae, and dynamic constraints, which are always of the form $p \supset [a]q$, p and q being arbitrary nonmodal wffs and a an atomic action. We implicitly assume an axiom of the form $\langle a \rangle$true for every atomic action a; this expresses the fact that the action a always terminates, though Q may only partially specify in what state a terminates. We also assume that Q is consistent.
R(u)	is a finite set of formulae called the <u>plan constraints</u>. Like the dynamic domain constraints, each of these is of the form $p \supset [u]q$ for nonmodal p and q. The symbol u is a distinguished atomic action not contained in $\mathcal{A}$.

A <u>solution</u> to a planning problem (V,Q,R(u)) is an expression α in the programming language $\hat{A}_V$ such that for every $r(\alpha)$ (obtained by substituting α for u in R), $Q \vdash r(\alpha)$. I.e., it is provable from the domain constraints Q that α satisfies all the plan constraints.[2] (Because of the termination constraints on the atomic actions, α is guaranteed to terminate. So in the language of program logic, we are talking about "total correctness.")

[1]For some applications it is desirable to constrain the programming language to use only a designated subset of the propositions as tests in conditionals. This requires a straightforward modification of our definition and will not be pursued here.

[2]Equivalently in semantic terms: Structures that satisfy the domain constraints also satisfy the α-instantiated plan constraints.

3.2. Finding Solutions

Having defined "*solutions*," we turn our attention to methods for discovering them. A natural way of organizing the search for solutions is to follow the syntactic structure of the programming language.

Recall that a program is either Λ, an atomic action a, or a composite of the form $\alpha;\beta$ or $t \rightarrow \alpha,\beta$ where α and β are programs and t is an atomic *proposition*. It will simplify the algorithm to consider only programs in a normal form, which we now define.

3.2.1. Normal Form for Conditional Plans

A program is in normal form if it consists of a sequence[3] of 0 or more atomic actions followed optionally by a conditional program both branches of which are in normal form, followed in turn by 0 or more further atomic actions. More formally, a program is in normal form if it can be written as $A_1;\ldots;A_n$, $n \geqslant 0$, with at most one A_i not atomic, in which case A_i is of the form $t \rightarrow B_1, B_2$ where both B_j are themselves in normal form.[4] The null sequence is identified with Λ, and we take $\Lambda;\alpha = \alpha;\Lambda = \alpha$.

We have not lost any essential solutions by insisting on this form since every C-program can be put into normal form by transforming the longest (length > 1) sequence of steps whose first and last steps are conditionals

[3]Since ";" is associative, we write sequences a;b;...;c without indicating order of association.

[4]Warren's method [17] for introducing conditionals produces plans of an even more restricted form: the *conditional* must be the <u>last</u> action in the sequence. I.e., a plan, once split, may never rejoin. This is not an essential limitation, but it introduces a somewhat greater degree of redundancy than our form. We note in passing that Warren's view of the conditional test as [illegible] action, has much in common with PDL's p? action.

into a single conditional as follows:

$(s \rightarrow A,B);\ \ldots\ ;\ (t \rightarrow C,D)$

$$\Rightarrow (s \rightarrow A;\ldots;(t \rightarrow C,D),\ \ B;\ldots;(t \rightarrow C,D))$$

and applying this transformation recursively to A, B, and the residual $\ldots;(t \rightarrow C,D)$.

3.2.2. An Algorithm

Suppose we are looking for a normal-form program α that satisfies one of the dynamic constraints $p \supset [\alpha]q$ in R. Consider the following cases corresponding to the possible forms of α :

1. $\alpha = \Lambda$. This is a solution if $Q \vdash p \supset q$.
2. $\alpha = a;\beta$ or $\alpha = \beta;a$ for some atomic action a. In the former case, α is a solution if $Q \vdash p/a \supset [\beta]q$, where p/a represents the strongest provable post-condition of p and a. Analogously, in the second case, α is a solution if $Q \vdash p \supset [\beta]a\backslash q$ where $a\backslash q$ is the weakest provable precondition of a and q. We call the former case "progression" and the latter "regression."
3. $\alpha = t \rightarrow \beta_1, \beta_2$. In this case, α is a solution if $Q \vdash p \wedge t \supset [\beta_1]q$ and $Q \vdash p \wedge \neg t \supset [\beta_2]q$.

We see that (1) defines success, (2) suggests forward and backward strategies for sequential steps, and (3) suggests a forward strategy for conditionals.

In addition, we see that there are several obvious ways to limit the search. First, if $p \supset p/a$, the forward search need not consider action a. (A special case of this arises when p/a = true.) Dually, if $a\backslash q \supset q$, the backward search need not consider a. (Here we have a special case when $a\backslash q$ = false.) These checks eliminate self-loops. We can eliminate cycling in the search space altogether if we are willing to pay the price of checking whether $p_i \supset p/a$ for any p_i in the leading chain of preconditions. Likewise we can check whether $a\backslash q \supset q_j$ for any q_j in the trailing chain of postconditions. Note also that if $p \supset t$ or $p \supset \neg t$, the forward conditional search involving t need not be pursued. p/a can never be false since this would imply failure of a to terminate, contradicting our assumptions about the domain constraints Q.

These observations lead immediately to the following non-deterministic algorithm for computing solutions for the single constraint $p \supset [\alpha]q$: (Multiple constraints will be discussed later.)

BIGRESSION[5] ALGORITHM

```
Assume p, q are not false.

Solve(p,q) = Bigress(p,q,$\Lambda$,$\Lambda$).

Bigress(pre,post,leader,trailer):

  IF  Q $\vdash$ pre $\supset$ post THEN RETURN( leader;trailer ).

  CHOOSE:

     CHOOSE <a, pre/a>  from LiveForward(pre):

         RETURN( Bigress( pre/a, post, leader;a, trailer ) )

     CHOOSE <a, a\post> from LiveBackward(post):

         RETURN( Bigress( pre,  a\post, leader,  a;trailer ) )

     CHOOSE t from NonTriv(pre):

         RETURN( leader; C ; trailer )

              where C = (t $\rightarrow$ Bigress(pre $\wedge$ t,  post,$\Lambda$,$\Lambda$),

                              Bigress(pre $\wedge\neg$ t, post,$\Lambda$,$\Lambda$))

LiveForward(p):
  IF $\emptyset \neq$ S
     where S = { <a, p/a> | a $\in \mathcal{A}$, Q $\nvdash$ p $\supset$ p/a}
  THEN RETURN( S )
  ELSE FAIL().

LiveBackward(q):
  IF $\emptyset \neq$ S
     where S = { <a, a\q> | a $\in \mathcal{A}$, Q $\nvdash$ a\q $\supset$ q}
  THEN RETURN( S )
  ELSE FAIL().
```

[5]"Bigression" stands for "bidirectional progression and regression."

```
NonTriv(p):
  IF $\emptyset \neq S$
     where $S = \{ t \mid t \in \mathcal{P},\ Q \not\vdash p \supset t,\ Q \not\vdash p \supset \neg t \}$
  THEN RETURN( S )
  ELSE FAIL().
```

The algorithm as presented finds solutions for a single plan constraint. However the extension to the general case is straightforward: To insure that all the plan constraints are met, a "Cartesian product" version of this algorithm must be run. A failure in any of the constraint components counts as failure and serves to prune that branch.

The bigression algorithm makes use of three additional auxiliary functions: "$Q\vdash$", "/", and "\". "$Q\vdash$" is a procedure which takes as input a nonmodal formulae p and decides whether p is provable from Q. If the static axioms, Q_s, are rich enough,[6] this check can be done using only nonmodal reasoning, i.e., ordinary propositional decision methods. The functions "/" (progression) and "\" (regression) are the subject of the next section.

3.2.3. Progression and Regression

Ideally, we would like p/a to compute the strongest postcondition of condition p and action a. Similarly, we would like a\q to compute the weakest precondition. [1, 15] In PDL, the weakest precondition of p and a can be expressed simply as [a]p, which is obviously the weakest formula implying "after a, p"! The strongest postcondition can be expressed using a "converse" operator which we have not described. (See [5].)

[6]Specifically, Q_s must generate all the nonmodal formulae generated by all of Q. In certain pathological cases, such as when Q contains a dynamic axiom of the form $p \supset [a]false$, Q would have to be extended to include extra static axioms, since $p \supset [a]false$ and $\langle a \rangle true$ together imply $\neg p$--a nonmodal formula derivable only through modal reasoning.

However, given the restricted form of our dynamic axioms, there will be no propositional formula provably equivalent to either of these modal formulae. On the other hand, we can effectively compute the weakest precondition <u>pre</u> and strongest postcondition <u>post</u> for which it is <u>provable</u> from Q that <u>pre</u> *implies* "after a, q" and p implies "after a, <u>post</u>." It is these propositional formulae that we label p/a and a\q.

The formula p/a is found by taking the conjunction of the set of formulae each of which is a disjunction of a set of q_i drawn from the "right hand side" of the dynamic axioms of Q ($p_i \supset [a]q_i$) such that the disjunction of the corresponding p_i's is implied by p. Dually, a\q is found by taking taking the disjunction of the set of formulae each of which is a conjunction of a set of p_i drawn from the "left hand side" of the dynamic axioms of Q ($p_i \supset [a]q_i$) such that that conjunction of the corresponding q_i's implies q.

Consider the following sample axioms:

$A \quad \supset [a]\ (B \vee C)$
$G \quad \supset [a]\ \neg B$
$(F \wedge E) \supset [a]\ D$

In this case, $a\backslash(C \vee D) = (A \wedge G) \vee (F \wedge E)$. The reason for this is that $(B \vee C)$ conjoined with $\neg B$ implies $(C \vee D)$, so the conjunction of the corresponding left-hand sides $(A \wedge G)$ is one disjunct of $a\backslash(C \vee D)$. Likewise, the formula D alone implies $(C \vee D)$, making the corresponding left-hand side $(F \wedge E)$ the second disjunct. These two cases exhaust the possibilities for getting $(C \vee D)$.

The reason why the formulas p/a and a\q defined in this way are not exactly equivalent to strongest postcondition and weakest precondition lies in the nature of our atomic actions. Briefly, in the context of programming languages one typically begins with primitives whose semantics are fully characterized and focuses on characterizing the derived operations (sequencing, etc.) [1] For example, the weakest precondition for the assignment primitive is given by

$$wp(\text{"}x := E\text{"}, P(x)) = P(E).$$

This equation asserts that the weakest precondition for condition P and action "x gets E" is precisely P with E substituted for x.

In our case, however, the primitive actions are specified only by axioms giving one-way implications. Thus, unless we make assumptions of a "non-monotonic" nature, we would generally be able to consistently add axioms that "weaken" the precondition or "strengthen" the postcondition of an action. Since "provably weakest" is unattainable, we make do with "weakest provable." *This does not affect the* completeness of the search algorithm, since we are only looking for programs which provably satisfy the specifications.

3.3. Hierarchical Planning

The key observation in extending the single-level algorithm to multi-level, hierarchical planning is that an atomic action at level k is a plan to be solved for at level k+1. This point of view is possible because of the way the planning problem was formalized. Specifically, an atomic action is described by a set of dynamic axioms in Q. Likewise, the desired program is described by a set of dynamic axioms in R. Since the same formal objects, namely sets of dynamic axioms, are involved in both cases, it is natural to assume as primitive some action with given properties at level k and then solve for a program having those properties at level k+1.

Formally, a hierarchical planning problem is a tree of single-level problems. If $<V_k = <\mathcal{P}_k, \mathcal{Q}_k>, Q_k, R_k(u_k)>$ is the problem at non-leaf node k, then node k has one successor for each $a_{k,i}$ in $\mathcal{Q}_k$, and that successor's problem has the form $<V_{k+1},Q_{k+1},Q'(a_{k,i})>$ where Q' denotes the subset of dynamic axioms of Q_k having the form $p \supset [a_{k,i}]q$. In other words, the domain constraints on the primitive "a" at level k become plan requirements at level k+1. A solution is a plan using the vocabulary of the leaf nodes that satisfies the requirements of the root node. I.e., α is a solution if it solves $<V_n,Q_n,R_1(u_1)>$. The propositional vocabulary and action vocabulary can change from level to level, provided the domain axioms have enough inferential structure to make the transfer from level to level meaningful.

Obviously, for any node k, only the successor nodes corresponding to actions actually used in the solution need be solved. Furthermore, the existence of a solution for each of these nodes guarantees the existence of an overall solution.

As with other hierarchic planners, the main benefit of levels in our approach is heuristic: The choice of intermediate vocabularies and domain axioms constitutes a choice of "planning islands." Any algorithm that tries to solve a problem by solving the nodes in the hierearchy is, in essence, searching for a plan constrained to go through the states defined by the intermediate actions' domain constraints. The main benefit of logic here is to define a reasonable relation between the levels, namely the relation: "correctly implements."

For a fixed determination of levels and a small number of actions it would be possible to precompute solutions to the subproblems, in which case after solving the problem at the top level, the system would act more like a compiler than a problem solver. In dynamic situations when the lower-level actions (in effect, the "tools" for solving the problem) are changing or when only a small number of actions are ever actually used, it seems more natural to solve subproblems as they arise.

4. Discussion

4.1. Modeling Actions: The Legacy of STRIPS Operators

Much of the research into the control of planning has been carried out in the STRIPS paradigm. [2, 6] In this approach, actions are regarded not as mappings from states to states, but rather as syntactic transformations of state-descriptions to state-descriptions, where state-descriptions are logical formulae. One consequence is the oft-cited benefit of not needing to mention the various "frame conditions," i.e. the properties which are invariant under an action.[7] Unfortunately, the need for operators to be sensitive to the syntax of state descriptions led researchers to consider only very simple state descriptions (e.g. sets of atomic propositions) and very simple transformations (e.g. add-lists and delete-lists).

As an example of an action that is difficult to specify with a single add-list/delete-list pair, consider the action <u>toggle</u> described by a pair of dynamic axioms:

$$\mathrm{On(light)} \supset [\ \mathrm{toggle(switch)}\ ]\ \neg\,\mathrm{On(light)}$$
$$\neg\,\mathrm{On(light)} \supset [\ \mathrm{toggle(switch)}\ ]\ \mathrm{On(light)}$$

Since the post-condition depends conditionally on the pre-condition, it cannot be determined isolation whether toggle adds or deletes the wff On(light). The same would hold true for actions with disjunctive post-conditions.

These possibilities notwithstanding, many planning systems do make the

[7]However, these invariants need not be as large an obstacle to practical implementation as is commonly supposed (see [10]).

assumption that the truth of a given atomic proposition in the state resulting from applying a sequence of operators is a determinate, calculable thing. Techniques which rely crucially on these assumptions are sometimes difficult to adapt to less constraining assumptions. We give two illustrations from NOAH. [11]

4.2. Nonlinear Planning: Problems with Partial Orders and Shuffles

The basic idea behind nonlinear planning is the following: To solve a conjunctive goal G1 & G2, find a sequence S1 = a;b;...;c which achieves G1 and another sequence S2 = d;e;...;f which achieves G2. Represent the overall plan as a network of partially ordered actions with S1 and S2 as parallel branches. Now use the "resolve conflicts critic" to detect interference between the plans and impose additional ordering constraints among the actions to rule out the interference. The network encodes the subset of possible shuffles of S1 with S2 which are believed to achieve the overall goal G1 & G2.

For the resolve conflicts critic to filter interference correctly, it must know what is true at each node of the network. Unfortunately, for nodes that occur after joins, what is true depends crucially on the ultimate linearization of the parallel branches. In the general case, the best that can be done is to represent the disjunction of the strongest postconditions of the alternative linearizations.[8] This requires considering the alternatives, of which there are $\binom{m+n}{m}$ where m and n represent the lengths of the action sequences in the two parallel branches. Since it is easy to imagine cases where resolve-conflicts criticism would be an expensive operation, the belief

[8]Actually, NOAH does not represent disjunctive postconditions-- which may explain why disjunctive goals are considered problematical.

that using a nonlinear strategy is computationally efficient seems to be grounded in the _empirical_ hypothesis that operators encountered in practice will permit easy detection of conflicts.

4.3. Hierarchical Planning: Problems with Heuristic Decompositions

The justification for partial orderings in NOAH is tied up with a desire not to _prematurely_ commit the system to a particular linear order of actions which, though seemingly correct at one level, may expand into incorrect plans at lower levels. This possibility can only arise, of course, if the relation between levels ("plan A achieves the same effect as action a") is not exact. However, such inexactness undermines the original rationale for hierarchic planning, namely factorization of complexity, since it destroys compositionality and requires that we check complex lower-level plans for "unexpected" _global_ interactions. Again, an empirical hypothesis is presumably invoked, namely that by some suitable metric, the plan comes "close" to implementing the abstract action. (It is not immediately obvious, though, what metric could be meaningful for the space in question.)

4.4. Some Benefits of Bigression

Some of the benefits of regression were first discussed by Waldinger [15] and appreciated by Warren. [16] These benefits are reaped dually by including progression, which completes the logical symmetry and allows bidirectional search. As we have described them, the progression and regression operations handle arbitrary boolean formulas, thus solving conjunctive and disjunctive goals as special cases of a more general strategy. Goals of maintenance and prevention can be incorporated into the algorithm as well by expressing as (nonmodal) wffs the condition to be maintained (m) and the condition to be prevented (v). Since the planning algorithm actually develops a descriptive wff (d) for each state reachable during plan execution, it is straightforward to add a check to the procedures LiveForward, LiveBackward, and NonTriv eliminating paths through states where $Q \vdash d \supset \neg m \vee v$.[9] This simple approach will work in situations where no dynamic replanning is anticipated; goals of maintenance and prevention involving execution monitoring, feedback and replanning, require more complex strategies.

ACKNOWEDGEMENTS

I have profited from discussions with Richard Waldinger, Vaughan Pratt, Kurt Konolige, Dave Wilkins, Jerry Hobbs, and Bob Moore.

[9] For a more thoroughgoing treatment of reasoning about processes with intermediate states, see [9] .

REFERENCES

1. Dijkstra, Edsger W. Guarded Commands, Nondeterminacy and Formal Derivation of Programs. Communications of the ACM 18, 8 (August 1975), 453-457.

2. Fikes, R.E. and N.J. Nilsson. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. Artificial Intelligence 2, 3-4 (Winter 1971), 189-208.

3. Harel, David. Lecture Notes in Computer Science. Volume 68: First Order Dynamic Logic. Springer-Verlag, 1979.

4. Hayes-Roth, B. and F. Hayes-Roth, S. Rosenschein, and S. Cammarata. Modeling Planning as an Incremental, Opportunistic Process. Proceedings of 6th International Joint Conference on Artificial Intelligence, Tokyo, August, 1979, pp. 375-383.

5. Litvintchouk, S.D. and V.R. Pratt. A Proof-Checker for Dynamic Logic. Proceedings of 5th International Joint Conference on Artificial Intelligence, Massachusetts Institute of Technology, August, 1977, pp. 552-558.

6. Nilsson, Nils J. Principles of Artificial Intelligence. Tioga Publishing Co., 1980.

7. Pratt, Vaughan R. Semantical Considerations on Floyd-Hoare Logic. Proceedings of the 17th IEEE Symposium on Foundations of Computer Science, October, 1976, pp. 109-121.

8. Pratt, Vaughan R. A Near-Optimal Method for Reasoning about Action. MIT/LCS/TM-113, Massachusetts Institute of Technology, September, 1978.

9. Pratt, Vaughan R. Six Lectures on Dynamic Logic. MIT/LCS/TM-117, Massachusetts Institute of Technology, December, 1978.

10. Rosenschein, Stanley J. Hierarchical Planning: Implementation Considerations. SRI Technical Report. Forthcoming.

11. Sacerdoti, Earl D. The Nonlinear Nature of Plans. Proceedings of 4th International Joint Conference on Artificial Intelligence, Tbilisi, Georgia, USSR, September, 1975, pp. 206-214.

12. Sacerdoti, Earl D. A Structure for Plans and Behavior. Elsevier, 1977.

13. Sussman, G.J. A Computer Model of Skill Acquisition. American Elsevier, New York, 1975.

14. Tate, Austin. Generating Project Networks. Proceedings of 5th International Joint Conference on Artificial Intelligence, Massachusetts Institute of Technology, August, 1977, pp. 888-893.

15. Waldinger, Richard. Achieving Several Goals Simultaneously. Technical Note 107, SRI International, July, 1975.

16. Warren, David H.D. WARPLAN: A System for Generating Plans. Department of Computational Logic Memo 76, University of Edinburgh, July, 1974.

17. Warren, David H.D. Generating Conditional Plans and Programs. Proceedings of Summer Conference on Artificial Intelligence and Simulation of Behavior, University of Edinburgh, July, 1976, pp. 344-354.